Lower Bounds for Four-player Pointer Jumping

David Blinn

Advisor: Amit Chakrabarti

Senior Honors Thesis

Submitted to the faculty in partial fulfillment of
the requirements for a major in Computer Science
Dartmouth College

June 1, 2006

Abstract

In four-player pointer jumping, players observe some of the edges in a directed graph consisting of five layers of nodes numbered $0 \dots 4$ where layer 0 has a single node and layer 4 has two nodes. All nodes except those in the last layer possess a single edge pointing to a node in the next layer. Players wish to calculate the node reached by following the path of edges from the first layer in the graph to the last.

In the general input model for pointer jumping, Player i observes all edges in the graph except those leading from layer $i-1$ to layer i. We give a lower bound for the one-way communication complexity of the four-player player pointer jumping function with the added restriction to the general input model that Player 3 does not see the edges from layer 1 to layer 2.

As an appendix, we provide observations obtained in our investigation into four-player pointer jumping function in the general model.

Acknowledgments

I am grateful to my committee members, professors Prasad Jayanti and Peter Winkler. I am also grateful to my advisor, professor Amit Chakrabarti, for his assistance, insights, and unshakable patience as he guided me through my thesis.

Contents

Chapter 1

Introduction

Computer science is a field that studies computational problems. Computer scientists may analyze the properties of such problems, attempt to solve them, or implement their solutions. A question central to all of these pursuits is: how challenging is it to solve a given computational problem? Complexity theory attempts to measure the difficulty of these problems in relation to all other such problems by grouping them into classes. These classes of problems may be organized on any of a variety of a problem's characteristics. Frequently studied characteristics include the amount of computation required to solve a problem (time complexity), or the amount of memory necessary to solve the problem (space complexity).

1.1 Communication Complexity

When multiple parties individually hold information that is mutually necessary to solve a computational problem, they must communicate in order to compute the correct solution. Communication complexity is the study of how much information these parties must send to each other. These problems, often referred to as communication games, are commonly constructed as follows: each party in a communication game is given an input or several inputs to a function, and the parties wish to know the function's output. No party observes all of the inputs to the function, and consequently they must send information to each other in order to compute its output.

1.2 The Two-Player Model

The most familiar model for communication games is the two-player model due to Yao. In this model, for a function $f : X \times Y \to Z$, there are two parties, Alice and Bob, assumed to have unlimited computing power. They are given inputs $x \in X$ and $y \in Y$ respectively, and they wish to compute the value $f(x, y)$. Alice and Bob communicate according to a fixed protocol composed of a series of stages in which one player sends a message in the form of a sequence of bits to the other player. At each stage, the protocol must determine if the communication game is over, and if not, specify which player speaks next. At any given stage, the protocol must determine the message of the player to speak based solely on the information available to that player. That is, the player's input and the messages sent thus far. At the termination of the protocol, at least one of the players must be able to compute $f(x, y)$.

The cost of such a protocol, $\mathcal{P}$, on input (x, y) is the total number of bits the players communicate to each other when running the protocol on this input. The cost of $\mathcal{P}$ is the maximum cost of $\mathcal{P}$ over all inputs (x, y). We say that the communication complexity of f is the minimum cost of any protocol $\mathcal{P}$ over all protocols that compute f. Note that if each player receives an input of n bits, a protocol can simply have one player send all n bits to the other, allowing him to compute f, placing a trivial upper bound on the communication complexity of the function. However, providing meaningful upper and lower bounds frequently relies on more sophisticated tools. We wish to examine lower bounds.

A key concept in lower bounding the communication complexity of functions in the two-player model is the notion of a combinatorial rectangle, often simply referred to as a rectangle. For $A \subseteq X$, $B \subseteq Y$, we call $R = A \times B \subseteq X \times Y$ as a rectangle. Rectangles are important to the study of communication protocols because the set of inputs that cause a protocol to compute any output form a combinatorial rectangle. Loosely, this is a consequence of the protocol having to specify a player's message without knowledge of the other player's input. As an example, suppose at a given stage, it is Alice's turn to speak and so far she has sent messages $a_1, ..., a_i$, and Bob has sent messages $b_1, ..., b_j$. Suppose that on this sequence of messages, Alice will respond with message a_{i+1} if she has as her input either x or x'. When Bob receives the message, he cannot distinguish whether Alice sent the message on either x or x'. Thus, if the protocol were to end here with output z and Bob sent his messages on input y, then the combinatorial rectangle $\{(x, y), (x'y)\}$ causes the protocol to compute z. Thus, if we determine that any protocol for f must partition $X \times Y$ into a minimum of t rectangles, the protocol must utilize at least t different sequences of messages. Since such sequences of messages are sent with bits and it takes at least $\log_2 t$ bits to describe the longest sequence, we can lower bound the complexity of f with $\Omega(\log t)$.

Variations on the basic two-party model sketched above include nondeterminism, randomized communication, probability distributions on the inputs, and more. For formal definitions and a more thorough introduction to all of these topics, refer to [5]. Another extension to the two-party model is the inclusion of more players in what are known as multiparty protocols.

1.3 Multiparty Protocols

Multiparty protocols include k players who wish to compute the value of a function $f : X_1 \times X_2 \times \ldots \times X_k \to Z$. There are variations on how players observe their inputs, but the one we wish to concern ourselves with is known as the *number on the forehead* model. Visually, one can think of player i in this model as having input i written on her forehead. That is, she observes all inputs except x_i. We interchangeably refer to this model in the context of four-player pointer jumping as the general input model.

Players communicate according to the *blackboard* model: that is, players can be thought of as writing their messages on a blackboard so that messages are observed by all players. The cost of a protocol on input $(x_1, x_2, \ldots, x_k)$ in this setting is defined as the number of bits that players write to the blackboard according to the protocol.

Proving lower bounds in this setting can be challenging because each player is aware of a greater portion of the input. Each player observes $k-1$ of k inputs, making the players' task of computing f easier so that players have less need to communicate. This makes the prover's task of lower bounding the necessary communication harder.

A combinatorial structure for multiparty protocols analogous to the function of rectangles for two-party protocols is the cylinder intersection.

Definition 1 *Let $S_j^{(i)} \subseteq X_j$. We define C_i to be a cylinder in the ith dimension if $C_i = S_1^{(i)} \times S_2^{(i)} \times \ldots \times X_i \times \ldots \times S_k^{(i)}$. That is, C_i is a cylinder in the ith dimension if membership in C_i does not depend on the ith coordinate. Thus, if $(x_1, \ldots, x_i, \ldots x_k) \in C_i$, then $\forall x_i' \in X_i, (x_1, \ldots, x_i', \ldots x_k) \in C_i$.*

We define C to be a cylinder intersection if it is the intersection of k cylinders in the dimensions 1 through k. That is, $C = \cap_{i=1}^{k} C_i$.

As with rectangle intersections, if any protocol that computes f must partition the input into at least t cylinder intersections, then we can lower bound the communication complexity of f with $\Omega(\log t)$. For more details concerning multiparty protocols and cylinder intersections, consult [5].

1.4 Rounds and One-way Protocols

We may constrain players in communication games to exchange only a limited number of *rounds* of messages. Rounds are defined in the two-party setting as the number of alternations between when Alice and Bob communicate [5]. For certain functions, restricting the number of rounds a protocol allows can dramatically increase the complexity. We define a round in multiparty protocols to be a group of stages in the protocol in which each player is allowed to speak at most once.

In one-round protocols, each player is allowed to speak only once. A special case of one-round protocols are one-way protocols, in which players speak only once in the order $P_1, P_2, ..., P_k$ with P_k writing the solution to the blackboard. One-way multiparty protocols are of special interest because of their relation to other types of complexity.

1.5 One-way Multiparty Protocols and ACC Circuits

A circuit is a directed acyclic graph with a set of inputs nodes, a set of gate nodes, and a set of output nodes. Input nodes are given boolean labels and are sources, meaning they have in-degree zero. Gate nodes are vertices labeled with boolean functions that take inputs equal to their fan-in, that is, their in-degree. Output nodes are sinks, that is, they have out-degree zero. The size of a circuit is the total number of gate nodes in the circuit. The depth of a circuit is the maximum number of edges between any input and output node [3]. For a more complete introduction to circuits, refer to [5].

The class of functions computable by unbounded fan-in circuits of polynomial size and bounded depth using the gates AND, OR, NOT, and MOD$_m$ for a fixed m is referred to as ACC. Any function in "ACC can be computed by a one-way protocol with polylogarithmic number of players and only polylogarithmic cost" ([2] citing a result of [4]). Providing a function computed with polylogarithmic number of players requiring more than polylogarithmic cost using a one-way or simultaneous protocol would show the function is not in ACC. No explicit function has yet been shown to not be in ACC; however, some functions exhibit characteristics that make them promising candidates. The pointer jumping function is one such candidate. This is because pointer jumping is known to be LOGSPACE-complete. Thus, proving pointer jumping is in ACC would show that all other functions in LOGSPACE are also in ACC. It seems doubtful that all of these functions actually belong to ACC because ACC is a complexity class in which membership is determined by the computational steps needed to solve a problem. In contrast, LOGSPACE is a complexity class that groups problems by the amount of space they require.

Chapter 2

Pointer Jumping

2.1 Definition

Several different definitions of pointer jumping are given in the literature, but the fundamental structure of the problem remains the same throughout these variations. We construct the basic framework for pointer jumping as follows: in a directed graph there are layers of nodes; each node has out-degree one with an edge to a node in the following layer; players receive a partial list of these edges, known as pointers, and wish to compute the node reached by k pointer jumps from a given starting node. The pointer jumping function, "simulates many naturally occurring functions, e.g. shifting, addressing, multiplication of binary numbers, convolutions, etc" [2]. We now lay out several flavors of the pointer jumping:

- Kushilevitz and Nisan [5] along with Nisan and Wigderson [8] envision the pointer jumping in the two-party model. Alice and Bob are each given a list of n pointers. Each pointer in one of these lists points to a pointer in the opposite list. The protocol is k-rounds and seeks to compute the output starting from a pointer in Alice's input.

- Damm *et al.* consider pointer jumping when there are k players. There is a starting node followed by k ordered layers, each containing n nodes. Players communicate according to a one-way protocol and observe inputs according to the conservative model (see Section 2.2) [2].

- Babai *et al.* consider the pointer jumping function as a special case of the more general composition function. We adopt their definition, given formally below.

Definition 2 (Babai et al. [1], Definition 9.10) *For a positive integer n, we set $[n] := \{1, \ldots, n\}$. Let $m_0, m_1, \ldots, m_k$ be positive integers. The k-party composition function takes as input a k-tuple $(f_1, \ldots, f_k)$ of functions, where $f_1 : [m_0] \to [m_1], f_2 : [m_1] \to [m_2], \ldots, f_k : [m_{k-1}] \to [m_k]$, and returns their composition, $f_k \circ f_{k-1} \circ \ldots \circ f_1$.*

Definition 3 (Babai et al. [1], Definition 9.10 cont'd.) *In the special case of the k-party composition function when $m_0 = 1$, $m_k = 2$, this function is boolean, and is called the k-party pointer jumping function.*

Lemma 1 *When the players speak in any order except the one-way order, the communication complexity of the k-party pointer jumping function is $O(\log n)$.*

Proof: If the players speak in any order except the one-way order, there must be some Player i who speaks before Player j where $i > j$. When it is Player i's turn to speak, Player i simply writes on the board the node reached by the first j pointer jumps. That is, Player i writes $f_i(f_{i-1}(\ldots(f_1(1))))) \in [m_i]$. Since $m_1 \leq n$, this requires at most $\log_2 n$ communication bits. Player j can see all functions $f_{i+1}, \ldots f_{k-1}, f_k$ and so Player j now has enough information to compute the output. Thus, the protocol uses $O(\log n)$ communication bits. []

2.1.1 The Relation to ACC

In Section 1.5 we related one-way protocols to ACC and observed that pointer jumping is a promising candidate for showing a function to not be in ACC. By Definitions 2 and 3 and the obsevations made in Section 1.5, in order to prove that pointer jumping is not in ACC, one would specifically need to show that the one-way or simultaneous communication complexity of pointer jumping with $k \geq \text{polylog}(n)$ is $\Omega(n^\epsilon)$ for a positive constant ϵ where $n = \max(m_1, m_2, \ldots, m_{k-1})$.

2.2 Lower Bounds for Multiparty Pointer Jumping in the Conservative Model

We now outline a result of Damm *et al.* for multiparty pointer jumping. For a thorough understanding of the result, refer to [2]. Damm *et al.* were curious to examine pointer jumping because of its association to the ACC complexity class 1.5. Consequently, they were interested in the one-way complexity of the pointer jumping function with large number of players. No lower bounds for pointer jumping with $k > 3$ had yet been shown, so to simplify their analysis, they examined the problem in the conservative model.

Recall the definition of pointer jumping used by Damm *et al.* which can be obtained from our Definition 3 by setting $m_1, \ldots, m_{k-1} = n$ and changing the last layer of nodes to contain n nodes. The conservative model restricts the input model by limiting Player i from seeing the inputs $f_1, \ldots, f_{i-1}$ that she would see in the general model. Instead, Player i observes the node reached by the first $i-1$ pointer jumps, that is, the value of $f_{i-1}(\ldots(f_1(1)))$. Player i continues to see inputs $f_{i+1}, \ldots, f_k$. Damm *et al.* first compute an upper bound on the communication complexity of pointer jumping when $k \leq \log^* n$, proving that there is a conservative protocol to compute the pointer jumping function using only $n \log^{k-1} n + O(n)$ communication bits (Damm *et al.* Theorem 4.1 [2]). They then consider lower bounds.

They provide two lower bounds, both of which make use of an analysis on the density of inputs on which the protocol will output the correct answer. Loosely, the idea is that the players' messages must partition the input space into cylinder intersections so that the average size is no larger than the density multiplied by the size of the total input space. These bounds are:

1. The conservative one-way communication complexity of the pointer jumping function for $k = o((n/\log n)^{1/3})$ is lower bounded by $\Omega(n/k^2)$ (Damm *et al.* Theorem 5.4 [2]). The basic technique they apply in arriving at this result is a reduction of conservative one-way protocols for k-party pointer jumping to a protocol for $(k-1)$-party pointer jumping. They iteratively perform the reduction until only one party remains to arrive at a contradiction.

2. The conservative one-way communication complexity of the pointer jumping function for any $k \leq \log^* n - \omega(1)$ is lower bounded by $\Omega((n \log^{k-1} n)(1 - o(1)))$ (Damm *et al.* Theorem 5.6 [2]). They prove this result using a counting argument to arrive at a contradiction based upon their density analysis.

2.3 Communication Complexity with Help

Most of the following section summarizes or quotes results from Babai *et al.* in [1]. We rely upon the communication with help framework established by Babai *et al.* in proving our main result.

Babai *et al.* introduce the notion of communication complexity with help for multiparty communication games in which the players wish to compute a function with output of b bits. They envision a Helper able to see all of the inputs that writes a help message on the board before the players begin their communication. If the Helper were to write a message of length b, the Helper could simply write the output. Therefore, they restrict the Helper to sending at most $r \leq b - 1$ bits (for more on communication with help, consult [1]).

Definition 4 (Babai et al. [1], Definition 1.2) *The communication complexity of f with help, denoted by $C^{help}(r, f)$ is the minimum cost of a protocol with r bits of help for f.*

Notation 1 (Babai et al. [1], Construction 6.1) *Let $B = \{0,1\}^b$, and let $f : X_2 \times \ldots \times X_k \to B$ be any function. We define $\tilde{f} : \{1, \ldots, b\} \times X_2 \times \ldots \times X_k \to \{0,1\}$ by $\tilde{f}(i, x_2, \ldots, x_k) = f(x_2, \ldots, x_k)_i$. In other words, Player 1's input specifies a single bit of the output of f.*

Let ${}^1C(\tilde{f})$ be the communication complexity of $\tilde{f}$ when Player 1 speaks first but never speaks again.

Lemma 2 (Babai et al. [1], Definition 6.2) ${}^1C(\tilde{f}) \geq \min\left(b, C^{help}(b-1, f)/b\right)$

Proof: "Suppose we are given a communication protocol that computes f, that begins with Player 1 sending at most $b - 1$ bits, but never speaking again. We use this protocol to construct a $(k-1)$-party protocol with help to compute the function f itself. In this protocol, on input $(x_2, \ldots, x_k)$, the Helper sends the same message Player 1 would send in the protocol for $\tilde{f}$. Players 2 through k now compute $\tilde{f}(i, x_2, \ldots, x_k)$ for each possible value of i, using the given protocol. After this, every bit of $f(x_1, \ldots, x_k)$ has been found, and at most $b^1C(\tilde{f})$ bits of communication have been used." [1]. (Note: we believe there is a typo in the last line this proof, and that it should actually read: "After this, every bit of $f(\underline{x_2}, \ldots, x_k)$ has been found...") The minimum is present in the bound because in the case where no protocol exists to compute f that begins with Player 1 sending at most $b-1$ bits but never speaking again, Player 1 must have sent at least b bits.

2.4 Lower Bounds on the Communication Complexity of the Bits of Hash Value Function

In formulating the communication with help framework, Babai *et al.* were inspired by a result of Nisan and Wigderson concerning the bits of hash value function [8]. The function can be defined as follows: $BHV : \{0,1\}^{2n} \times \{0,1\}^n \times \{0,1\}^{\log n}$ The first set is a 2-universal family of hash function (a hash family where all functions in the family have equal probability of mapping a given input to any given output) from $\{0,1\}^n \to \{0,1\}^n$. The second set consists of inputs to the hash function, and the last set consists of indices into the output. Thus, $BHV(x, y, z)$ computes $x(y)_z$, that is, the zth bit of $x(y)$.

Nisan and Wigderson give a lower bound of $\Omega(\sqrt{n})$ for the one-way communication complexity of the bits of has value function [8]. They prove the bound using an analysis of probabilities conditioned on the messages the players send then invoking the Leftover Hash Lemma of Mansour *et al.* [6]. We had a somewhat difficult time understanding their argument and found the recasting of the bound in Babai *et al.* to be clearer [1].

Babai *et al.* obtain the same result by placing the problem in the formal communication with help framework. They invoke the same Leftover Hash Lemma and then examine the output of the hash function before it is indexed using multicolor discrepancy [1]. For more on multicolor discrepancy, we refer the curious reader to [1] and [5].

2.5 Lower Bounds on the Communication Complexity of Three-player Pointer Jumping

Citing an unpublished proof by Wigderson, Babai *et al.* provide an $\Omega(\sqrt{n})$ lower bound on the one-way communication complexity with an analysis utilizing the communication with help framework. Some of their techniques served as inspiration for the tools we employ in our analysis. They first relate three-player pointer jumping to two-player composition with help. They then provide a proof via a shifting argument allowing them to assume that Player 3, the player whose message indicates the value computed by the protocol, outputs a message of a certain form. This section of the proof included an observation concerning near-Hamming balls that that we were unable to relate to their overarching analysis of pointer jumping and composition. Finally, they provide a counting argument analyzing the number of inputs on which the protocol computes the correct answer over sets of inputs corresponding to the messages sent by Player 2 in order to arrive at the bound.

In their proof of the bound, Babai *et al.* use crucially the fact that the sets causing Player 2 to send a given message are rectangles. In four-player pointer jumping, the analogous sets are cylinder intersections, although we introduce a simplification to the input model to allow us to assume these sets are cylinders rather than cylinder intersections.

Chapter 3

Lower Bounds on the Communication Complexity of Four-player Pointer Jumping

We examine the one-way communication complexity of the four-party pointer jumping function when $n = m_3 = m_2 = (m_1)^2$ (see Definition 3). Further, we limit the input model by restricting Player 3 from seeing the input (f_2). In this model:

1. Player 1 sees (f_2, f_3, f_4)
2. Player 2 sees (f_1, f_3, f_4)
3. Player 3 sees (f_1, f_4)
4. Player 4 sees (f_1, f_2, f_3)

This makes the pointer jumping function slightly harder for the players to compute, and thus it is slightly easier to lower bound its communication complexity. Though the bound for four-party pointer jumping in the number on the forehead model is the more interesting problem, we believe our techniques may be an important step toward finding a lower bound in the number on the forehead input model.

Theorem 1 (Main result) *For the four-player pointer jumping function in the above restricted model where* $n = m_3 = m_2 = (m_1)^2$*, the one-way communication complexity is* $\Omega(\sqrt{n})$*.*

Building upon the techniques presented in [1], our proof proceeds by relating the the four-party pointer jumping function to a three-party composition function with help, and then showing that in order to solve the problem with fewer than some amount of communication, at least one of several conditions, each of which would lead to a contradiction, must be true.

The three-party composition function we wish to consider takes as input the three tuple (f_2, f_3, f_4), and outputs their composition: $f_4 \circ f_3 \circ f_2$. Note that because f_2 takes m_1 inputs

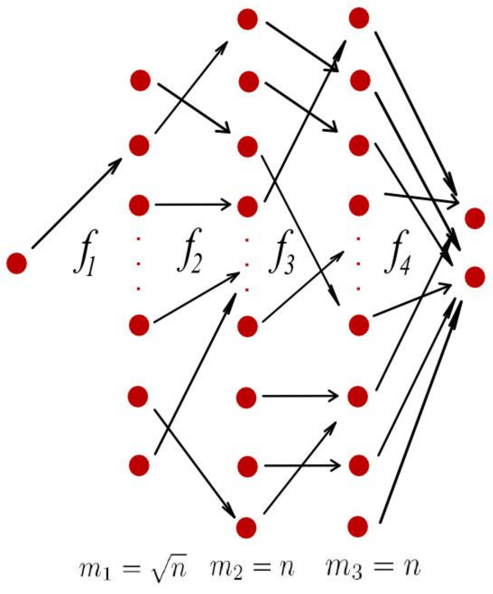

Figure 3.1: The graph view of four-player pointer jumping.

and f_4 has two possible output values, it takes m_1 bits to describe a composition of this sort. There is no longer an input f_1. Communication occurs with Player 1 acting as Helper, speaking first by sending the help message, and then communication proceeds in a one-way fashion.

Suppose we are given a protocol of cost less than $\sqrt{n}/32$ for the four-party pointer jumping problem. We can use this protocol to construct a new protocol for the three-party composition function with help that uses at most $n/32$ communication bits. We can do this as follows. Player 1 sends as a help message the same message he would have sent in the protocol for four-party pointer jumping. Then, for each $1 \leq i \leq m_1$, Player 2 sends the message he would have sent if $f_1(1) = i$ given Player 1's message in the original protocol. Likewise, Player 3 then sends for each $1 \leq i \leq m_1$ the message she would have sent if $f_1(1) = i$ given Player 1 and 2's messages in the original protocol. Player 4 now has enough information to compute $f_4 \circ f_3 \circ f_2(i)$ for each $i \in [m_1]$, and thus can output the desired composition. We will limit the Helper to sending fewer than $\sqrt{n}/8$ bits. If the Helper must send $\sqrt{n}/8$ bits or more, the original protocol must have used at least $\Omega(\sqrt{n})$ bits of communication.

We now present a shifting argument to reduce the size of the set of Player 4's output. This technique is similar to that presented in [1] for a similar purpose. Recall that in the three-party composition function, Player 4 receives inputs (f_2, f_3) and outputs a function from $[m_1] \to [m_4]$. Thus, we can describe Player 4's output with the function $b(f_2, f_3)$: $[m_2]^{[m_1]} \times [m_3]^{[m_2]} \to [m_4]^{[m_1]}$. The number of such functions b is $\left(m_4^{m_1}\right)^{m_2^{m_1} m_3^{m_2}}$. The shifting will reduce the size of the output space we must consider to $m_4^{m_3}$.

Before presenting the shifting argument, we first introduce some notation that we will use in the proof of Lemma 3 and the subsequent remark.

Notation 2 *Let m_1, m_2, m_3, m_4 be positive integers. α, β, and $\gamma, \hat{\gamma}, \gamma'$ are functions: $\alpha \in [m_2]^{[m_1]}$, $\beta \in [m_3]^{[m_2]}$, and $\gamma, \hat{\gamma}, \gamma' \in [m_4]^{[m_3]}$. $S, \hat{S}, S' \subseteq [m_4]^{[m_3]}$. When we draw α or β, at random, α is drawn with uniform probability from $[m_2]^{[m_1]}$ and β is drawn with uniform probability from $[m_3]^{[m_2]}$. In the proof of Lemma 3, γ is drawn with uniform probability from S. Let $b, \hat{b}$ be functions such that $b, \hat{b} : [m_2]^{[m_1]} \times [m_3]^{[m_2]} \to [m_4]^{[m_1]}$. Finally, $p(b, S, \alpha, \beta) = \Pr_\gamma[b(\alpha, \beta) = \gamma \circ \beta \circ \alpha]$.*

Lemma 3 (Shifting Lemma, modeled on Babai et al. [1], Lemma 9.16) *For all pairs (b, S), for all $\hat{\gamma}$, there exists $\hat{S}$ such that $|\hat{S}| = |S|$ and for all α, β, $p(b, S, \alpha, \beta) \leq p(\hat{b}, \hat{S}, \alpha, \beta)$ where $\hat{b}$ denotes the function $\hat{\gamma} \circ \beta \circ \alpha$.*

We give a construction modeled on the shifting construction in [1] to prove the Lemma. The outer loop of the construction obeys the following invariant: at the end of the loop, for every α, β, $p(b, S, \alpha, \beta)$ either increases or remains the same.

```
for $j$:=1 to $m_3$ do
    $S' := \emptyset$;
    for $\gamma \in S$ do
        Let $\gamma'$ be defined by $\gamma'(i) := \begin{cases} \hat{\gamma}(j) if i = j \\ \gamma(i) otherwise \end{cases}$;
        if $\gamma' \notin S$ then
            add $\gamma'$ to $S'$;
        else
            add $\gamma$ to $S$;
        end
    end
    $S := S'$;
    Redefine $b$ so that, for all $\alpha, \beta$ $\begin{cases} b(\alpha,\beta)(i) = \hat{\gamma}(j) \text{ when } \beta(\alpha(i)) = j \\ b(\alpha,\beta)(i) \text{ is unchanged otherwise} \end{cases}$;
end
```

Observe that at the end of each iteration of the outer loop, for each γ that was in S, either a γ or a γ' is added to S'. Thus, every time the construction replaces S with S', the size of S remains the same. To see that at the end of each iteration of the outer loop, for every α, β, $p(b, S, \alpha, \beta)$ either increases or remains the same, note the following for each iteration of the outer loop:

1. For a given $j \in [m_3]$, we modified b so that the function it outputs behaves the same as the the composition $\hat{\gamma} \circ \beta \circ \alpha$ does for all (α, β) when the composition $\beta \circ \alpha$ would output j

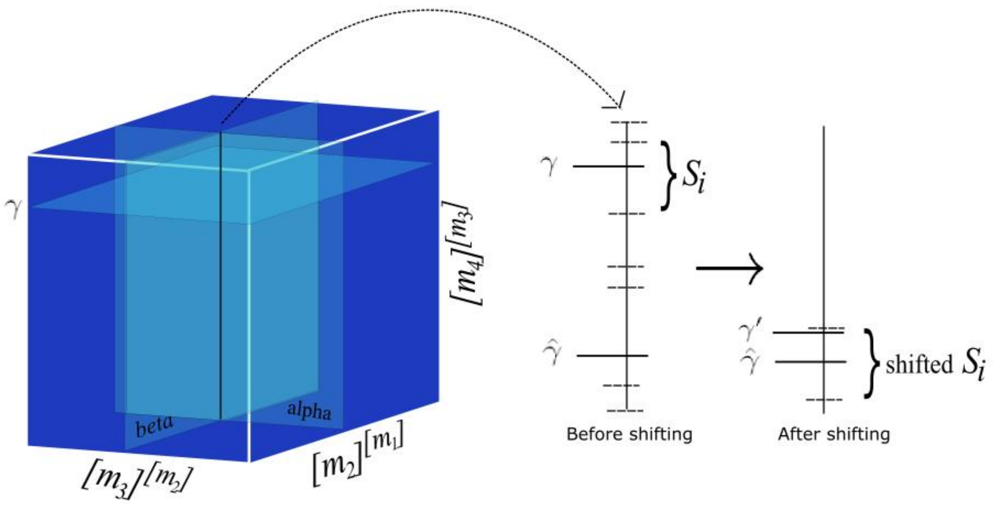

Figure 3.2: α and β specify a vertical line to which we may apply the shift.

2. For a given $j \in [m_3]$, the inner loop either changed functions $\gamma \in S$ to behave as $\hat{\gamma}$ does on j or left them as they were. No γ is changed on any other coordinate, so the number of $\gamma \in S$ that behave as $\hat{\gamma}$ does on *all* coordinates either increases or remains the same at the end of each iteration of the outer loop.

This proves the Lemma. Note that this construction would still behave as desired if, outside of the loops, we set for all α, β, $b(\alpha, \beta)(i) = \hat{\gamma}(\beta(\alpha(i)))$. We have written the construction as above so that it obeys the invariant used in proving the Lemma. []

Remark 1 (The Application of the Shift) *Suppose we have a partition $\Phi = Y_1 \cup \ldots \cup Y_q$ of $[m_2]^{[m_1]} \times [m_3]^{[m_2]} \times [2]^{[m_3]}$ where q is a positive integer. Note that the code given in the proof of the Shifting Lemma (Lemma 3) does not actually shift sets of the form $Y \subseteq [m_2]^{[m_1]} \times [m_3]^{[m_2]} \times [2]^{[m_3]}$, but instead shifts sets of the form $S \subseteq [2]^{[m_3]}$. Observe that the pair (α, β) specify a vertical line in the cuboid $[m_2]^{[m_1]} \times [m_3]^{[m_2]} \times [2]^{[m_3]}$ (Figure 3). Taking this vertical line in the partition Φ specifies a partition of $[2]^{[m_3]}$ of the form $\Pi = S_1 \cup \ldots \cup S_p$ where p is a positive integer. With this in mind, we apply the shift to the space $[m_2]^{[m_1]} \times [m_3]^{[m_2]} \times [2]^{[m_3]}$ by iterating through all pairs $(a, b) \in [m_2]^{[m_1]} \times [m_3]^{[m_2]}$ and shift the sets in the corresponding partition with respect to $\hat{\gamma}$.*

We now return to consider the composition function itself. In the protocol with help, denote the message h as the help message sent by Player 1, s as the message sent by Player 2, and t as the message sent by Player 3. We now define a set.

Definition 5 $Y_{h,s,t} = \{(f_2, f_3, f_4)$: *Player 2 sends s given h, and Player 3 sends t given (h, s)*$\}$.

The inputs to the three-party composition function can be thought of as composing a cuboid in three dimensions with the f_4 axis pointing toward the top of the page and the f_2

axis pointing into the page as in Figure 3. The sets $Y_{h,s,t}$ form cylinders within this cuboid. (In the general input model, the equivalent sets would be cylinder intersections). Note that these cylinders partition the cuboid on fixed h. By the Shifting Lemma (Lemma 3), we can refer to Player 4's output for an input $(f_2, f_3, f_4) \in Y_{h,s,t}$ after shifting by $g_{h,s,t} \in [m_4]^{[m_3]}$. We now consider the consequences of shifting these zones according to the shifting construction given in the proof for the Shifting Lemma (Lemma 3) all with respect to a fixed $g \in [m_4]^{[m_3]}$. For simplicity, we now refer to any $g_{h,s,t}$ by g.

Definition 6 *The Hamming distance of functions $a, b : C \to D$ in the same family is the number of inputs on which a and b produce different outputs. We denote the Hamming distance of a, b as $\mathrm{dist}(a, b)$. We call a **near** to b if $\mathrm{dist}(a, b) < |C|/4$. We call a **far** from b if $\mathrm{dist}(a, b) \geq |C|/4$.*

In the proofs of Lemmas 4, 5, and 6 below, we use the following conventions. Fix a help message h. We say the shifting construction given in the proof of the Shifting Lemma (Lemma 3) maps $(f_2, f_3, f_4) \in Y_{h,s,t}$, to (f_2, f_3, f'_4). Further, throughout the rest of the proof, we define N_h to be the number of help messages used by the protocol and we define $N_{s,t}$ and $N_{h,s,t}$ to be the number of message pairs (s, t) and number of message triples (h, s, t) respectively.

Lemma 4 (Similar to Babai et al. [1], Lemma 9.18)

$$|\{(f_2, f_3, f_4) : f'_4 \textit{ is near } g]\}| \leq m_2^{m_1} m_3^{m_2} m_4^{m_3} 2^{-m_3/6} \times N_{s,t}$$

Proof: Enumerate the zones $Y_{h,s,t}$ from $Y_1, \ldots, Y_{N_{s,t}}$. Then:

$$\begin{aligned} \Pr_{f_2,f_3,f_4}[f'_4 \text{ is near } g] &= \Pr_{f_2,f_3,f_4}[(f_2, f_3, f_4) \in Y_1 \wedge f'_4 \text{ is near } g] + \ldots \\ &+ \Pr_{f_2,f_3,f_4}[(f_2, f_3, f_4) \in Y_{N_{s,t}} \wedge f'_4 \text{ is near } g]. \end{aligned}$$

Observe that for all g, the number of functions f'_4 near to g is less than or equal to $\sum_{i=0}^{m_3/4} \binom{m_3}{i} (m_4 - 1)^i$. From binary entropy, we have that $\sum_{i=0}^{m_3/4} \binom{m_3}{i} < 2^{m_3 H(1/4)}$ where H is the binary entropy function. From [1], we have that $H(1/4) < 5/6$. Thus, the number of functions f_4 near to $g < 2^{5m_3/6}(m_4 - 1)^{m_3/4}$. For $m_4 = 2$, we can simplify this to $2^{5m_3/6}$. Thus:

$$\Pr_{f_2,f_3,f_4}[f'_4 \text{ is near } g \mid (f_2, f_3, f_4) \in Y_i] \leq \frac{2^{5m_3/6} m_2^{m_1} m_3^{m_2}}{|Y_i|}.$$

Plugging in, we see that:

$$\sum_{i=1}^{N_{s,t}} \Pr_{f_2,f_3,f_4}[(f_2, f_3, f_4) \in Y_i] \Pr_{f_2,f_3,f_4}[f'_4 \text{ is near } g \mid (f_2, f_3, f_4) \in Y_i] =$$

$$\sum_{i=1}^{N_{s,t}} \frac{|Y_i|}{m_2^{m_1} m_3^{m_2} 2^{m_3}} \frac{2^{5m_3/6} m_2^{m_1} m_3^{m_2}}{|Y_i|} = 2^{-m_3/6} \times N_{s,t}.$$

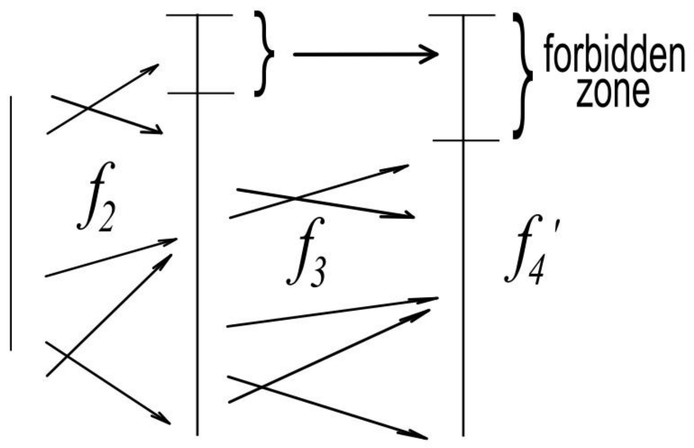

Figure 3.3: The "forbidden zone" consists of all inputs i to f_4' where $f_4'(i) \neq g(i)$. If f_3 maps fewer than 1/8 of its inputs to this zone, we call it *bad*.

Thus:

$$\Pr_{f_2,f_3,f_4}[f_4' \text{ is near } g] \leq 2^{-m_3/6} \times N_{s,t}$$

. And because we have exactly $m_2^{m_1}$ choices for f_2, $m_3^{m_2}$ choices for f_3 and $m_4^{m_3}$ choices for f_4, we can use this probability to bound the size of the set considered in the statement of the Lemma above by a simple multiplication. This completes the proof. []

We now consider those points $(f_2, f_3, f_4) \in Y_{h,s,t}$ that the shifting construction maps to points (f_2, f_3, f_4') where f_4' is *far* from g.

Definition 7 *We call f_3 **bad** with respect to (f_4', g) if f_3 maps fewer than $m_2/8$ inputs to points in $[m_3]$ on which f_4' and g disagree. That is, f_3 is bad with respect to (f_4', g) if $|\{i : i \in [m_2], g(f_3(i)) \neq f_4'(f_3(i))\}| \leq m_2/8$. We call f_3 **good** with respect to (f_4', g) otherwise.*

We choose the nomenclature bad and good because on a bad f_3, Player 4 has a high likelihood of outputting the correct answer over all f_2. We, as the provers, wish to make it as difficult as possible for the players to compute the correct answer, and so for us, this is bad.

We now observe a property of the shifting construction given in the proof of the Shifting Lemma (Lemma 3) when it is applied to a cylinder $Y_{h,s,t}$.

For $f_4 \in [m_4]^{[m_3]}$, we define P_{f_4} to be the plane in the input cuboid corresponding to f_4. In set notation, $P_{f_4} = \{((\alpha, \beta, f_4)) : \alpha \in [m_2]^{[m_1]}, \beta \in [m_3]^{[m_2]}\}$. For $f_3 \in [m_4]^{[m_3]}$, $f_4 \in [m_4]^{[m_3]}$, we define L_{f_3,f_4} to be the line corresponding to f_3 on the plane corresponding to f_4 corresponding to f_4. In set notation, $L_{f_3,f_4} = \{(\alpha, f_3, f_4) : \alpha \in [m_2]^{[m_1]}\}$.

Observe that because neither Player 2 nor 3 knows the input f_2, all cylinders $Y_{h,s,t}$ extend fully across the f_2 axis of the input cuboid. Mathematically, for $\alpha_1, \alpha_2 \in [m_2]^{[m_1]}$, $(\alpha_1, f_3, f_4) \in Y_{h,s,t} \leftrightarrow (\alpha_2, f_3, f_4) \in Y_{h,s,t}$. A little thought shows that as a result of this property, any line L_{f_3,f_4} will lie entirely inside the same cylinder $Y_{h,s,t}$ and as a result it will be shifted contiguously so that all points on the line are shifted to a new line $L_{f_3,f_4'}$. Thus, we may speak of the shift as shifting lines rather than merely shifting points.

Definition 8 *We call $L_{f_3,f_4'}$ a **bad line** if f_3 is bad with respect to (f_4', g). We call $L_{f_3,f_4'}$ a **good line** otherwise.*

Lemma 5

$$|\{(f_2, f_3, f_4) : f_4' \textit{ is far from } g \ \wedge L_{f_3,f_4'} \textit{ is a bad line}\}| \leq \\ m_2^{m_1} m_3^{m_2} m_4^{m_3} e^{-m_2/32} \times N_{s,t}$$

Proof: We prove this result using the Chernoff bound.

Fix a plane $P_{f_4'}$ where f_4' is far from g. Let

$$p_i = \Pr_{f_3}[f_4'(f_3(i)) \neq g(f_3(i))].$$

Because f_4' is far from g, for all $i \in [m_2]$, $p_i \geq 1/4$. Let X_i be an indicator random variable for whether a randomly selected f_3 maps i to one of the points on which f_4' and g disagree. Then $\Pr[X_i = 1] = p_i$, and $\Pr[X_i = 0] = 1 - p_i$. We define $X = \sum_{i=0}^{m_2} X_i$. Clearly, $E(X) \geq m_2/4$.

Applying the Chernoff bound (see [7]), we see that:

$$\Pr[X < (1 - 1/2)\frac{m_2}{4}] \leq e^{-m_2(1/4)\frac{(1/2)^2}{2}} = e^{-m_2/32}.$$

This gives us the probability that a line $L_{f_3,f_4'}$ is bad for fixed f_4' far from g. Note, however, that the shifting construction may shift more than one line to the same $L_{f_3,f_4'}$ (Figure 3). At most one line may be shifted to this position per cylinder $Y_{h,s,t}$. Thus, for any (f_3, f_4) where f_4' is far from g, we can bound the probability that $L_{f_3,f_4'}$ will be bad with $e^{-m_2/32} \times N_{s,t}$. Since each line contains $m_2^{m_1}$ points and there are $m_3^{m_2}$ choices for f_3 and $m_4^{m_3}$ choices for f_4, we have that:

$$|\{(f_2, f_3, f_4) : f_4' \text{ is far from } g \wedge L_{f_3,f_4'} \text{ is a bad line}\}| \leq m_2^{m_1} m_3^{m_2} m_4^{m_3} e^{-m_2/32} \times N_{s,t}$$

and this proves the bound. []

Lemma 6

$$|\{(f_2, f_3, f_4) : f_4' \textit{ is far from } g \wedge L_{f_3,f_4'} \textit{ is a good line} \wedge \\ f_4' \circ f_3 \circ f_2 = g \circ f_3 \circ f_2\}| \leq m_2^{m_1} m_3^{m_2} m_4^{m_3} e^{-m_1/8}$$

Proof: We bound this quantity using a technique similar to that used by Babai *et al.* in their Lemma 9.16 [1].

Consider any line L_{f_3,f_4} such that $L_{f_3,f_4'}$. Observe that on this line, $\text{dist}(f_4' \circ f_3\ , g \circ f_3) \geq m_2/8$. Thus, if f_2 maps any of its inputs to these $m_2/8$ points of disagreement, Player 4 will

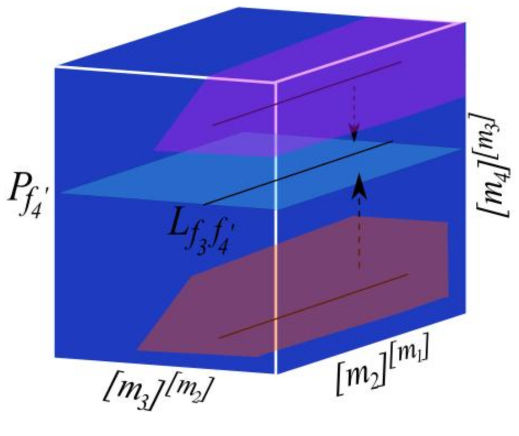

Figure 3.4: The shifting construction may shift more than one line to the same $L_{f_3,f_4'}$. The colored regions represent cylinders.

output the incorrect answer (and, moreover, (f_2, f_3, f_4) will not belong to the set considered in the statement of the Lemma). Since f_2 varies freely on the line in consideration and takes m_1 inputs,

$$\Pr_{f_2}[f_4' \circ f_3 \circ f_2 = g \circ f_3 \circ f_2] \leq \left(1 - \frac{m_2/8}{m_2}\right)^{m_1} \leq e^{-m_1/8}.$$

This gives the probability Player 4 outputs the corrrect answer for a random input on a good line. Since the number of good lines L_{f_3,f_4} is at most the total number of lines, $m_3^{m_2} m_4^{m_3}$, and each such line contains exactly $m_2^{m_1}$ points,

$$\begin{aligned} |\{(f_2, f_3, f_4) : L_{f_3,f_4'} \text{ is a good line } \wedge f_4' \circ f_3 \circ f_2 = g \circ f_3 \circ f_2\}| \\ \leq m_2^{m_1} m_3^{m_2} m_4^{m_3} e^{-m_1/8}. \end{aligned} \tag{3.1}$$

Observe that the set considered in the statement of the Lemma is a subset of the set considered in equation 3.1, and so this proves the Lemma. []

Theorem 2 (Restatement of the main theorem) *For the four-player pointer jumping function in the restricted model where*

1. *Player 1 sees* (f_2, f_3, f_4)
2. *Player 2 sees* (f_1, f_3, f_4)
3. *Player 3 sees* (f_1, f_4)
4. *Player 4 sees* (f_1, f_2, f_3)

and $n = m_3 = m_2 = (m_1)^2$, *the one-way communication complexity is* $\Omega(\sqrt{n})$.

Proof: We begin the proof by defining several sets.

Definition 9 $X_{h,s,t} = \{(f_2, f_3, f_4)$: *Player 1 sends help message* h, *Player 2 sends* s, *and Player 3 sends* $t\}$.
$Z_{h,s,t} = \{(f_2, f_3, f_4) : g \circ f_3 \circ f_2 = f_4 \circ f_3 \circ f_2\} \cap Y_{h,s,t}$

Observe that $X_{h,s,t} \subseteq Z_{h,s,t}$. To see why this is the case, note that because the protocol is deterministic, on all inputs that cause (h, s, t) to be sent, Player 4 must output the correct answer $b_{h,s,t}(f_2, f_3) = f_4 \circ f_3 \circ f_2$. However, if we fix a help message h, it is not required by the protocol that an input that causes the players to output $(s, t, b_{h,s,t}(f_2, f_3) = f_4 \circ f_3 \circ f_2)$ given h would actually cause the Helper to send h.

Since the sets $X_{h,s,t}$ partition the input space $[m_2]^{[m_1]} \times [m_3]^{[m_2]} \times [m_4]^{[m_3]}$,

$$\sum_{h,s,t} |X_{h,s,t}| = m_2^{m_1} m_3^{m_2} m_4^{m_3}.$$

Fix a function $g \in [m_4]^{[m_3]}$ to shift with. Observe that we can partition the sets $Z_{h,s,t}$ into sets $Z^{\text{near}}_{h,s,t}$ and $Z^{\text{far}}_{h,s,t}$ that contain those points in $Z_{h,s,t}$ that are mapped to f'_4 near to and far from g respectively. Further note that we can partition the sets $Z^{\text{far}}_{h,s,t}$ into sets $Z^{\text{far,bad}}_{h,s,t}$ and $Z^{\text{far,good}}_{h,s,t}$ that contain lines $L_{f_3,f_4} \in Z^{\text{far}}_{h,s,t}$ that are mapped to lines L_{f_3,f'_4} that are respectively bad and good.
By Lemmas 4, 5, and 6:

1. $\sum_{s,t} |Z^{near}_{h,s,t,g}| \leq m_2^{m_1} m_3^{m_2} m_4^{m_3} 2^{-m_3/6} \times N_{s,t}$
2. $\sum_{s,t} |Z^{far,bad}_{h,s,t,g}| \leq m_2^{m_1} m_3^{m_2} m_4^{m_3} e^{-m_2/32} \times N_{s,t}$
3. $\sum_{s,t} |Z^{far,bad}_{h,s,t,g}| \leq m_2^{m_1} m_3^{m_2} m_4^{m_3} e^{-m_1/8}$.

Thus, we have:

$$\begin{aligned} m_2^{m_1} m_3^{m_2} m_4^{m_3} &\leq \sum_{h,s,t} |Z_{h,s,t}| \\ &\leq \sum_{h} m_2^{m_1} m_3^{m_2} m_4^{m_3} \left((2^{-m_3/6} \times N_{s,t}) + (e^{-m_2/32} \times N_{s,t}) + e^{-m_1/8} \right). \end{aligned}$$

Plugging in $m_1 = \sqrt{n}, m_2 = n, m_3 = n$:

$$\begin{aligned} 1 &\leq \sum_{h} (2^{-n/6} \times N_{s,t}) + \sum_{h} (e^{-n/32} \times N_{s,t}) + \sum_{h} e^{-\sqrt{n}/8} \\ &= 2^{-n/6} \times N_{h,s,t} + 2^{(-(n/32) \log_2 e)} \times N_{h,s,t} + 2^{-\sqrt{n}/8 \log_2 e} \times N_h \\ &< 2^{-(n/32) \log_2 e + 1} \times N_{h,s,t} + 2^{-\sqrt{n}/8 \log_2 e} \times N_h \,. \end{aligned}$$

In order for the inequality to hold, either the number of message tuples (h, s, t) is at least $2^{n/32 \log_2 e}$ or the number of help messages h is at least $2^{\sqrt{n}/8 \log_2 e - 1}$. By the arguments we presented when we constructed the protocol for the composition function and Lemma 2, the original protocol must have used at least $\Omega(\sqrt{n})$ bits of communication. This completes the proof. []

Chapter 4

Appendix: The General Input Model and Uniform Shifting

Note that in the general input model, we cannot guarantee that lines remain contiguous after shifting as in the restricted input model. This is because the sets $Y_{h,s,t}$ on fixed h form cylinder intersections rather than merely cylinders. As a result, we cannot apply the techniques we used in the restricted model directly to the general model and obtain a meaningful bound. In our consideration of techniques to translate our result for four-player pointer jumping in the restricted setting where Player 3 does not see f_2 to the general input model, we observed and proved an interesting property about the distribution of points after they have been shifted according to the construction given in the proof of the Shifting Lemma (Lemma 3). We give this result here.

Recall Notation 2. We reuse some of this notation with the amendment that m_4 explicitly equals 2 in the proof of the Uniform Shifting Lemma (Lemma 7).

Notation 3 *Let m_1, m_2, m_3, m_4 be positive integers and $m_4 = 2$. $\alpha, \beta,$ and $\gamma, \hat{\gamma}, \gamma'$ are functions: $\alpha \in [m_2]^{[m_1]}$, $\beta \in [m_3]^{[m_2]}$, and $\gamma, \hat{\gamma}, \in [m_4]^{[m_3]}$.*

Lemma 7 (Uniform Shifting Lemma) *Let $\Pi = S_1 \cup \ldots \cup S_p$ where p is a positive integer be any partition of $[2]^{[m_3]}$. Let $\gamma, \hat{\gamma}$ be chosen uniformly at random and let S_i be the set in the partition where $\gamma \in S_i$. If we shift S_i with respect to $\hat{\gamma}$, the modified γ has uniform probability in $[2]^{[m_3]}$.*

Proof: In proving this Lemma, we first make a claim about the outcome of shifting coordinate j of γ.
Claim: Let γ, $\hat{\gamma}$, and S_i be defined as above. When we shift coordinate j of the functions in S_i with respect to $\hat{\gamma}$ the modified γ is uniform random.
Proof: Observe that a uniform random function γ in the hypercube $[2]^{[m_3]}$ can be decomposed into two uniform random variables: one, which we will call γ_{-j}, describing all co-ordinates but the jth one, and one, which we will call γ_j, describing just the jth coordinate. Note that γ_j is a 1-bit variable. In (Figure 4), when $\gamma_j = 1$, γ belongs to the

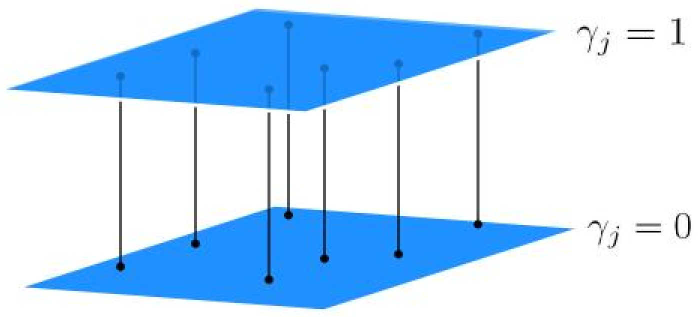

Figure 4.1: The decomposition of γ into γ_{-j} and γ_j. A line represents a value of γ_{-j}. A point at the end of a line represents a value of γ.

hypercube represented by the top plane, and when $\gamma_j = 1$, γ represented by the bottom plane. We refer to the point opposite γ in the other hypercube, that is, the point described by γ_{-j} and $\neg(\gamma_j)$, as $\check{\gamma}$.

The shift does not alter γ_{-j} and so γ_{-j} remains uniform random after the shift. Thus, showing that the the shift results in uniform random γ_j is sufficient to prove the claim. We can break down the outcome of the shift on γ_j into two complementary cases, both of which produce uniform random γ_j:

1. $\check{\gamma} \notin S_i$: In this case, the shift sets $\gamma_j = \hat{\gamma}_j$. Because $\hat{\gamma}_j$ is uniform random, after the shift, γ_j is uniform random.

2. $\check{\gamma} \in S_i$: In this case, the shift does not alter the value of γ_j. Because γ_j was uniform random before the shift and was not altered, it is uniform random after the shift.

This completes the proof of the claim.

Notice that the shift itself works by shifting coordinates $j = 1, j = 2, \ldots j = m_3$. This completes the proof of the lemma. []

Returning to the context of four-player pointer jumping in the general model, suppose we select uniformly at random a point in the input cuboid (f_2, f_3, f_4) and shift it with respect to a randomly selected function $g \in [2]^{[m_3]}$. Let (f_2, f_3, f_4') be the point after it has been shifted. Using the Uniform Shifting Lemma (Lemma 7), we had hoped to eliminate the dependence of f_4' on (f_2, f_3). This would have allowed us to use a Chernoff bound and counting argument similar to those used in Lemmas 5 and 6 respectively to bound the number of inputs on which Player 4 produces the correct answer on fixed help message h (Note that this discussion only considers the far inputs. A little investigation shows that the bound for near inputs does not need adjustment in the general input model). However, (f_2, f_3) are necessary to determine the cylinder intersection in which (f_2, f_3, f_4) lies. The cylinder intersection is in turn used by the shift in determining f_4'. Thus, f_4' and (f_2, f_3) are *not* independent.

These tools may be helpful in generating a lower bound for pointer jumping with $k > 3$ in the general input model, but unfortunately we have thus far been unable to apply them in a meaningful fashion. We present them here to the reader in order to provide a full disclosure into the state of our research and hope that they provide some insight into the problem.

Chapter 5

Concluding Remarks

5.1 Remarks

Our ambition when we began our investigation of four-player pointer jumping was to provide a substantive lower bound on the one-way communication complexity in the number on the forehead model. Further along into our research we had hoped to use the techniques we established in the restricted model as a starting point and the Universal Shifting Lemma as a stepping stone to reach a bound in the general input model. That we did not fully accomplish our goal in the time we had for this attempt should not discourage further attempts; rather we hope that the results we did achieve will make the steps that remain in order to show a bound for the general model that much easier.

Additionally, we believe our result for four-player pointer jumping even in the limited model is not without significance. No meaningful lower bounds have yet been shown for pointer jumping with $k > 3$ in the number on the forehead model, and we have taken a healthy step toward that. Although Damm *et al.* proved lower bounds for pointer jumping with large numbers of players in the conservative model, our input model is in some ways less restrictive than theirs because we only restrict the inputs viewed by a single player whereas they restrict the inputs observed by all players except for a single player.

5.2 Open Problems

The logical next step in investigating the pointer jumping problem is to complete the formulation of a bound on the one-way communication complexity with four players in the general input model. If techniques similar to ours are used in the formulation of such a bound, it would be interesting to examine how many players the result would extend to. A polylogarithmic number of players with greater than polylogarithmic cost is, of course, the ultimate goal (see Section 1.5) but it remains to be seen for how many players the techniques presented here remain viable. Observe that in our proof of Lemma 5, we employed a constant fractional reduction in the number of points on which the composition $f_4' \circ f_3$ disagreed with $g \circ f_3$ from the number of points on which f_4' disagreed with g in order to invoke the Chernoff bound. Repeated application of this technique would lead to an extension to a logarithmic number of players.

Bibliography

[1] Laszlo Babai, Thomas P. Hayes, and Peter G. Kimmel. The cost of the missing bit: Communication complexity with help. *Combinatorica*, 21:455–488, Oct 2001.

[2] Carsten Damm, Stasys Jukna, and Jiri Sgall. Some bounds on multiparty communication complexity of pointer jumping. *Comput. Complex.*, 7(2):109–127, 1998.

[3] Oded Goldreich, Vered Rosen, and Alon Rosen. Introduction to complexity theory lecture 20: Circuit depth and space complexity. Available online at www.wisdom.weizmann.ac.il/˜oded/PS/CC/l27.ps, 1999.

[4] Johan Hastad and Mikael Goldmann. On the power of small-depth threshold circuits. *Computational Complexity*, 1:113 – 129, June 1991.

[5] Eyal Kushilevitz and Noam Nisan. *Communication Complexity.* Cambridge University Press, 1997.

[6] Yishay Mansour, Noam Nisan, and Prasoon Tiwari. The computational complexity of universal hashing. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (May 14–16 1990: Baltimore, MD)*, pages 235–243, New York, NY, 1990. ACM Press.

[7] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[8] Noam Nisan and Avi Wigderson. Rounds in communication complexity revised. *SIAM Journal on Computing*, 22:211–219, 1 1993.